

An FPTAS for Approximating a Sum of Random Variables ^{*}

Jian Li [†]

Tianlin Shi [‡]

Abstract

Given n independent random variables X_1, X_2, \dots, X_n and an integer C , we study the fundamental problem of computing the probability that the sum $X = X_1 + X_2 + \dots + X_n$ is at most C . We assume that each random variable X_i is implicitly given by an oracle which, given a input value k , returns the probability $X_i \leq k$. We give the first deterministic fully polynomial-time approximation scheme (FPTAS) to estimate the probability up to a relative error of $1 \pm \epsilon$. Our algorithm is based on the idea developed for approximately counting knapsack solutions in [Gopalan et al. FOCS11].

1 Introduction

We study the following fundamental problem: The input consists of n independent (not necessarily identically distributed) random variables X_1, \dots, X_n and an integer C . ¹ Let their sum be $X = X_1 + X_2 + \dots + X_n$. Our task is to compute the following probability value

$$F(C) = \Pr \left[\sum_{i=1}^k X_i \leq C \right].$$

Assumptions: We assume that all random variables are discrete and the support of X_i , denoted as supp_i , consist of only integers. We also assume that each random variable X_i is implicitly given by an oracle (denoted as \mathcal{O}_i) which, given a input value k , returns the probability $X_i \leq k$. Note that this assumption is weaker than assuming the explicit representation of X_i (listing the probability mass at every point), since we need to use binary search with $|\text{supp}_i| \log |\text{supp}_i|$ calls to the oracle to construct the explicit representation of X_i .

It is well known that computing $F(C)$ is #P-hard (see e.g., [9]). The hardness of computing $F(C)$ has an essential impact in the area of stochastic optimization as many problems generalize and/or utilize this basic problem in one way or another, thus inheriting the #P-hardness. Although we can sometimes use for example the linearity of expectation to bypass the difficulty of computing $F(C)$, more than often no such simple trick is applicable, especially in the context of risk-aware stochastic optimization where people usually pay more attention to the tail probability than the expectation.

Despite the importance of the problem, surprisingly, no approximation algorithm with provable multiplicative factor is known. We note that we can easily obtain an additive PTAS for this problem via the Monte-Carlo method: One generates K independent samples $X^{(i)}$, $i = 1, 2, \dots, K$ of X and use the empirical sum

$$\tilde{F} = \frac{1}{K} \sum_{i=1}^K I(X^{(i)} \leq C) \quad (1)$$

^{*}Institute for Interdisciplinary Information Sciences, Tsinghua University, China.

[†]Email: lijian83@mail.tsinghua.edu.cn

[‡]Email: stl501@gmail.com

¹ So, the running time of a polynomial time algorithm should be a polynomial of $\log C$.

where $I(\cdot)$ is the indicator function. By standard Chernoff bound, one can easily see that with $K = \text{poly}(1/\epsilon)$ samples, the estimation is within an additive error ϵ from the true value with a constant probability (see e.g., [14]). To get a reasonable multiplicative approximation factor, we need to set the value of ϵ at the order of $F(C)$, so the number of samples needs to be $\text{poly}(1/F(C))$, which can be exponentially large, when $F(C)$ is exponentially small ².

In this paper, we propose a *fully polynomial approximation scheme* (FPTAS) for counting $F(C)$. For ease of notation, we use $(1 \pm \epsilon)F(C)$ to denote the interval $[(1 - \epsilon)F(C), (1 + \epsilon)F(C)]$. Recall that we say there is an FPTAS for the problem, if for any fixed constant $\epsilon > 0$, the algorithm can produce a value \tilde{F} such that $\tilde{F} \in (1 \pm \epsilon)F(C)$ in polynomial (w.r.t. the length of the binary encoding of the input) time (See e.g., [14]).

Theorem 1.1. *We are given n independent random variables X_1, \dots, X_n distributed on support $[N]$, a positive integer C , a fixed positive constant ϵ ($\epsilon > 0$). Suppose there is an oracle for each X_i , which, upon two input integers (n_1, n_2) , returns the value $\Pr[n_1 \leq X_i \leq n_2]$. There is an FPTAS for estimating $\Pr[\sum_{i=1}^n X_i \leq C]$ and the running time is $O(\frac{n^3}{\epsilon^2} \log(\frac{1}{q})^2 \log(nN))$ where $q = \prod_i \min_{x \in \text{supp}(X_i)} \Pr[X_i = x]$.*

In the above theorem, we assume the binary coding of the value $\Pr[X_i = x]$ is polynomially bounded for any $x \in \text{supp}(X_i)$. Hence, $\log q$ is polynomial. Note that in the theorem, we can assume $N \leq C + 1$ by moving all probability mass exceeding $C + 1$ to point $C + 1$ without affecting the final answer.

1.1 Related Works

There is a large body of work on estimating or upper/lower-bounding the distribution of the sum of independent random variables. See e.g., [1, 18, 12, 3, 13]. Those works are based on analytic numerical methods (e.g., Edgeworth expansion, saddle point method) which either require specific families of distributions and/or do not provide any provable multiplicative approximation guarantees.

Our problem is a generalization of the counting knapsack problem. For the counting knapsack problem, Morris and Sinclair [15] obtained the first FPRAS (fully-polynomial randomized approximation scheme). Dyer [4] provided a completely different FPRAS based on dynamic programming. The first deterministic FPTAS is obtained by Gopalan et al. [7].

Our problem is also closely related to the threshold probability maximization problem (see a general formulation in [11]). In this problem, we are given a ground set of items. Each feasible solution to the problem is a subset of the elements satisfying some property (this includes problems such as shortest path, minimum spanning tree, and minimum weight matching). Each element b is associated with a random weight X_b . Our goal is to find a feasible set S such that $\Pr[\sum_{b \in S} X_b]$ is maximized. There is a large body of literature on the threshold probability maximization problem, especially for specific combinatorial problems and/or special distributions. For example, Nikolova, Kelner, Brand and Mitzenmacher [17] studied the corresponding shortest path version for Gaussian, Poisson and exponential distributions. Nikolova [16] extended this result to an FPTAS for any problem with Gaussian distributions, if the deterministic version of the problem has a polynomial time algorithm. The minimum spanning tree version with Gaussian distributed edges has also been studied in [5]. For general discrete distributions, Li and Deshpande [10] obtained an additive PTAS if the deterministic version of the problem can be solve exactly in pseudopolynomial time. Very recently, Li and Yuan [11] further generalized this result to the class of problems for which the multi-objective deterministic version admits a PTAS.

Our problem is also closely related to the fixed set version of the stochastic knapsack problem. In this problem, we are given a knapsack of capacity C and a set of items with random sizes and profits. Their goal

² In certain application domains such as risk analysis, small probabilities (often associated with catastrophic losses) can be very important.

is to find a set of items with maximum total profit subject to the constraint that the overflow probability is at most a given parameter γ . Kleinberg, Rabani and Tardos [9] first considered the problem with Bernoulli-type distributions and provided a polynomial-time $O(\log 1/\gamma)$ -approximation. Better results are known for specific distributions, such as exponentially distributions [6], Gaussian distributions [8, 16]. For general discrete distributions, bi-criteria additive PTASes³ are known via different techniques [2, 10, 11].

2 Algorithm

Our FPTAS is based on dynamic programming. In Section 2.1, we provide the recursion of the dynamic program, which is largely based on the idea developed in [7], with some necessary adaptations. However, since the support of each random variable can be exponentially large, it is not immediate clear how the recursion can be implemented efficiently given the oracles. In Section 2.2, we address this issue.

2.1 The Dynamic Program

We first notice that $\Pr[\sum_{j=1}^i X_j \leq C]$ is a nondecreasing function of C . We consider an inverse function $\tau(i, a) : \{0, 1, \dots, n\} \times \mathbb{R} \rightarrow \mathbb{R} \cup \{\pm\infty\}$, which is defined to be the smallest C such that the probability $\Pr[\sum_{j=1}^i X_j \leq C] \geq a$. It is easy to see that $\tau(i, a)$ is nondecreasing in a . We need the following simple lemma. We omit the proof, which is straightforward.

Lemma 2.1.

$$\Pr[\sum_{j=1}^n X_j \leq C] = \max\{a : \tau(n, a) \leq C\}.$$

The following recursion is very important to us.

Lemma 2.2. *Suppose $d_i \in \text{supp}(X_i)$, $p(d_i) = \Pr[X_i = d_i]$, $\gamma : \text{supp}(X_i) \rightarrow [0, 1]$. We have the following recurrence*

$$\tau(i, a) = \min_{\gamma} \max_{d_i} \{\tau(i-1, \gamma(d_i)) + d_i\} \quad (2)$$

subject to

$$\sum_{d_i \in \text{supp}(X_i)} \gamma(d_i) p(d_i) \geq a \quad (3)$$

Intuitively, suppose that $\gamma(d_i)$ represents the probability $\Pr[\sum_{j=1}^{i-1} X_j \leq C' - d_i]$ for some C' . (3) would enforce that $\Pr[\sum_{j=1}^{i-1} X_j \leq C'] = \sum_{d_i \in \text{supp}(X_i)} p(d_i) \gamma(d_i) \geq a$. The formal proof is as follows.

Proof. We first prove that for every choice of $\gamma : \text{supp}(X_i) \rightarrow [0, 1]$ (a list of $\gamma(x_i)$ for $x_i \in \text{supp}(X_i)$), the quantity $C' = \max_{d_k \in \text{supp}(X_i)} \{\tau(i-1, \gamma(d_k)) + d_k\} \geq \tau(i, a)$. Due to the independence, We can see that

$$\Pr[\sum_{j=1}^i X_j \leq C'] = \sum_{d_k \in \text{supp}(X_i)} \Pr[\sum_{j=1}^{i-1} X_j \leq C' - d_k] \Pr[X_i = d_k]$$

By the definition of C' , we know that $C' - d_k \geq \tau(i-1, \gamma(d_k))$ for any $d_k \in \text{supp}(X_i)$, thus $\Pr[\sum_{j=1}^{i-1} X_j \leq C' - d_k] \geq \gamma(d_k)$. So

$$\Pr[\sum_{j=1}^i X_j \leq C'] \geq \sum_{d_k \in \text{supp}(X_i)} \gamma(d_k) p(d_k) \geq a$$

³ The overflow probability constraint may be violated by an additive factor ϵ for any constant $\epsilon > 0$.

Algorithm 1 The Dynamic Program for Computing $\Pr[\sum_i X_i \leq C]$

```

1: Set  $T[1, j] = \tau(1, Q^{-j})$  for all  $j \geq 0$ .
2: Set  $Q = 1 + \frac{\ln(1+\epsilon)}{n+1}$  and  $s = \lceil \log_Q \frac{1}{q} \rceil$ .
3: for  $i = 2 \rightarrow k$  do
4:   for  $j = 0 \rightarrow s$  do
5:     Set  $T(i, j) = \min_{\lambda} \max_{d_i} T(i-1, j + \lfloor \log_Q(\frac{p(d_i)}{\lambda(d_i)}) \rfloor) + d_i$ 
6:   end for
7: end for
8: Let  $j' = \min\{j : T[n, j] \leq C\}$ . Output  $F' := Q^{-j'+1}$ .
```

Second, we prove that there exists a choice of γ' such that $C' = \max_{d_k} \{\tau(i-1, \gamma'(d_k)) + d_k\} \leq \tau(i, a)$. Let

$$\gamma'(d_k) = \Pr\left[\sum_{j=1}^{i-1} X_j \leq \tau(i, a) - d_k\right] \quad (4)$$

It is easy to see that γ' satisfies (3). By definition, $\tau(i-1, \gamma'(d_k)) \leq \tau(i, a) - d_k$ for all $d_k \in \text{supp}(X_i)$, which leads to $\max_{d_k} \{\tau(i-1, \gamma'(d_k)) + d_k\} \leq \tau(i, a)$. This completes the proof. \square

For ease of notation, we let $\lambda(d_i)a = \gamma(d_i)p(d_i)$. The recursion can be written as:

$$\tau(i, a) = \min_{\lambda} \max_{d_i} \left\{ \tau(i-1, \frac{\lambda(d_i)}{p(d_i)}a) + d_i \right\} \quad \text{subject to} \quad \sum_{d_i \in \text{supp}(X_i)} \lambda(d_i) \geq 1 \quad (5)$$

It is not clear how recursion (5) can be efficiently executed since the second argument a is a continuous variable. We use the following way to discretize it: Let $Q = 1 + \frac{\ln(1+\epsilon)}{n+1}$. Note that Q is slightly larger than 1. We use q to denote $\prod_{i \in [n]} \min_{d_i \in \text{supp}(X_i)} \Pr[X_i = d_i]$, which is clearly a lower bound of the answer. Let $s = \lceil \log_Q \frac{1}{q} \rceil = O(\frac{n}{\epsilon} \log_2 \frac{1}{q})$.

We define recursively the function $T : \{0, 1, \dots, n\} \times \{0, 1, \dots, s\} \rightarrow \mathbb{R} \cup \{\pm\infty\}$ as follows:

$$T(i, j) = \min_{\lambda} \max_{d_i} T\left(i-1, j + \lfloor \log_Q\left(\frac{p(d_i)}{\lambda(d_i)}\right) \rfloor\right) + d_i \quad \text{subject to} \quad \sum_{d_i \in \text{supp}(X_i)} \lambda(d_i) \geq 1 \quad (6)$$

Comparing (6) and (5), the similarity suggests that $T(i, j)$ is the approximate version of $\tau(i, Q^{-j})$. The next lemma formalizes this idea.

Lemma 2.3. *For all $i \in \{0, 1, \dots, n\}$ and $j \in \{0, 1, \dots, s\}$, we have*

$$\tau(i, Q^{-j}) \leq T(i, j) \leq \tau(i, Q^{-(j-i)}) \quad (7)$$

Proof. We prove this by induction. The base case is trivial by the definition of $T(0, j)$. Now assume for the statement is true for $i-1$ and all $j \in \{0, 1, 2, \dots, s\}$. We prove it is also true for i and all $j \in \{0, 1, 2, \dots, s\}$. By induction hypothesis, we have that

$$T(i-1, \lfloor j + \log_Q \frac{p(d_i)}{\lambda(d_i)} \rfloor) \leq \tau(i-1, Q^{-(\lfloor j + \log_Q \frac{p(d_i)}{\lambda(d_i)} \rfloor - i + 1)}) \leq \tau(i-1, \frac{\lambda(d_i)}{p(d_i)} Q^{-(j-i)}) \quad \text{and}$$

$$T(i-1, \lfloor j + \log_Q \frac{p(d_i)}{\lambda(d_i)} \rfloor) \geq \tau(i-1, Q^{-\lfloor j + \log_Q \frac{p(d_i)}{\lambda(d_i)} \rfloor}) \geq \tau(i-1, \frac{\lambda(d_i)}{p(d_i)} Q^{-j}).$$

Algorithm 2 Efficient Implementation of Recursion (6)

```
1: Set Left = 0, Right =  $kN$ .
2: while Right > Left do
3:   Set  $T' = \lfloor (\text{Left} + \text{Right})/2 \rfloor$ .
4:   for  $m = 1 \rightarrow s$  do
5:     Let  $p(j, m) = \Pr[X_i \in (T' - T(i-1, m-1), T' - T(i-1, m))]$  (via oracle  $\mathcal{O}_i$ ).
6:   end for
7:   if  $\sum_{m=1}^s Q^{m-j} p(j, m) \geq 1$  then
8:     Set Right =  $T'$ 
9:   else
10:    Set Left =  $T' + 1$ .
11:   end if
12: end while
13: Set  $T(i, j) = T'$ .
```

Taking the maximum over d_i and the minimum over λ do not change the direction of the inequalities. Combining the above inequalities with (5), we complete the proof. \square

With this lemma, we can approximate the recursion 5 by solving the recursion 6. The pseudo-code of the dynamic program is provided in Algorithm 1. It is not hard to show the output of the algorithm is a good approximation of the true probability.

Lemma 2.4. *The output F' of the algorithm is a $(1 \pm \epsilon)$ -approximation of $F(C) = \Pr[\sum_i^n X_i \leq C]$.*

Proof. From the choice of j' , we know that $T(n, j') \leq C < T(n, j' - 1)$. According to Lemma 2.3, we have $\tau(n, Q^{-j'}) \leq C < \tau(n, Q^{n-j'+1})$. Consequently $F = \Pr[\sum_{i=1}^n X_i \leq C] \in [Q^{-j'}, Q^{n-j'+1}]$. By outputting $F' = Q^{-j'+1}$, the approximation ratio can be bounded as $1 - \epsilon < Q^{-(n+1)} \leq \frac{F'}{F} \leq Q^{n+1} = 1 + \epsilon$. \square

2.2 An Efficient Implementation using Binary Search

In this subsection, we show how to implement the recursion (6) in polynomial time. Suppose we have already computed $T(i', j')$ for all $i' < i$ and $0 \leq j' \leq s$ and are computing the value $T(i, j)$. Our approach is based on a binary search on the range of $T(i, j)$. For every guess T' , we can decide whether $T(i, j) \leq T'$ efficiently by using the criterion in Lemma 2.5. The pseudocode is provided in Algorithm 2.

Lemma 2.5. *We define $\lambda'(d_i) = \max\{\lambda(d_i) \mid T(i-1, j + \lfloor \log_Q(\frac{p(d_i)}{\lambda(d_i)}) \rfloor) + d_i \leq T'\}$. Then, $T(i, j) \leq T'$ if and only if $\sum_{d_i \in \text{supp}(X_i)} \lambda'(d_i) \geq 1$.*

Proof. Suppose $\sum_{d_i \in \text{supp}(X_i)} \lambda'(d_i) \geq 1$. This means there exists a choice of λ such that

$$\max_{d_i} T(i-1, j + \lfloor \log_Q(\frac{p(d_i)}{\lambda(d_i)}) \rfloor) + d_i \leq T',$$

and therefore $T(i, j) \leq T'$ by (6). On the other hand, suppose $\sum_{d_i \in \text{supp}(X_i)} \lambda'(d_i) < 1$. Clearly, any other choice of $\lambda(d_i)$ should be smaller than $\lambda'(d_i)$ in order to obtain $T(i, j) \leq T'$, which is not possible. \square

From Lemma 2.5, we can see that if the $|\text{supp}(X_i)|$ is bounded by a polynomial, we can decide whether $T(i, j) \leq T'$ in polynomial time. However, $|\text{supp}(X_i)|$ can be exponentially large and only an oracle \mathcal{O}_i is provided. We resolve this issue by noticing that the support of each random variable can be divided into s segments such that the sum of $\lambda'(d_i)$ values over each segment can be computed efficiently. In particular,

we divide $[N]$ into segments $(T' - \infty, T' - T(i - 1, 0)] \cap [N], (T' - T(i - 1, 0), T' - T(i - 1, 1)] \cap [N], \dots, (T' - T(i - 1, s - 1), T' - T(i - 1, s)] \cap [N]$. In each segment, $\lambda'(d_i)$ can be computed according to the following lemma.

Lemma 2.6. *Suppose $d_i \in (T' - T(i - 1, m - 1), T' - T(i - 1, m)]$ for some $m \in [1, s]$. Then*

$$\lambda'(d_i) = Q^{m-j} p(d_i)$$

Proof. For any $d_i \in (T' - T(i - 1, m - 1), T' - T(i - 1, m)]$, by definition of λ' , we have that

$$j + \log_Q \left(\frac{p(d_i)}{\lambda'(d_i)} \right) = m.$$

So $\lambda'(d_i) = Q^{m-j} p(d_i)$ and we complete the proof. \square

As a result, it is sufficient to ask the \mathcal{O}_i for the probability value $\Pr[X_i \in (T' - T(i - 1, m - 1), T' - T(i - 1, m))]$ and we can compute $\sum_{d_i > T' - T(i - 1, m - 1)}^{T' - T(i - 1, m)} \lambda'(d_i)$ simply by

$$\sum_{d_i > T' - T(i - 1, m - 1)}^{T' - T(i - 1, m)} \lambda'(d_i) = \sum_{d_i > T' - T(i - 1, m - 1)}^{T' - T(i - 1, m)} Q^{m-j} \Pr[X_i \in (T' - T(i - 1, m - 1), T' - T(i - 1, m))].$$

Lemma 2.7. *Suppose each query to the oracle takes $O(1)$ time. The optimization with respect to λ, d_i in equation (6) runs in time $O((\frac{n}{\epsilon})^2 (\log \frac{1}{q})^2 \log_2(nN))$*

Proof. The binary search takes $\log_2(nN)$ iterations, while in each iteration we compute λ' for at most s segments. For each segment, we query an oracle \mathcal{O}_i , which takes one unit of time, to obtain the probability of the random variable X_i being in the segment. Q^{m-j} can be computed in advance and its contribution to running time is neglected. So the implementation of each recursion step runs in $O(s \log_2(nN) = O(\frac{n}{\epsilon} \log(\frac{1}{q}) \log(nN)))$. \square

Since the number of goal states in the dynamic program is $O(ns)$, according to Lemma 2.7, the total running time is $O(\frac{n^3}{\epsilon^2} \log(\frac{1}{q})^2 \log(nN))$. This completes the proof of Theorem 1.1.

References

- [1] George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.
- [2] A. Bhargat, A. Goel, and S. Khanna. Improved approximation results for stochastic knapsack problems. In *ACM-SIAM Symposium on Discrete algorithms*, 2011.
- [3] Henry E Daniels. Tail probability approximations. *International Statistical Review/Revue Internationale de Statistique*, pages 37–48, 1987.
- [4] Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 693–699. ACM, 2003.
- [5] S. Geetha and KPK Nair. On stochastic spanning tree problem. *Networks*, 23(8):675–679, 1993.
- [6] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *Annual Symp. on Foundations of Computer Science*, page 579, 1999.

- [7] Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Stefankovic, Santosh Vempala, and Eric Vigoda. An fptas for # knapsack and related counting problems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 817–826. IEEE, 2011.
- [8] V. Goyal and R. Ravi. Chance constrained knapsack problem with random item sizes. *Operation Research Letter*, 2009.
- [9] J. Kleinberg, Y. Rabani, and É. Tardos. Allocating bandwidth for bursty connections. In *ACM Symp. on Theory of Computing*, page 673, 1997.
- [10] J. Li and A. Deshpande. Maximizing expected utility for stochastic combinatorial optimization problems. In *Annual Symp. on Foundations of Computer Science*, 2011.
- [11] Jian Li and Wen Yuan. Stochastic combinatorial optimization via poisson approximation. In *ACM Symp. on Theory of Computing*, 2013.
- [12] Robert Lugannani and Stephen Rice. Saddle point approximation for the distribution of the sum of independent random variables. *Advances in applied probability*, pages 475–490, 1980.
- [13] Neelesh B Mehta, Jingxian Wu, Andreas F Molisch, and Jin Zhang. Approximating a sum of random variables with a lognormal. *Wireless Communications, IEEE Transactions on*, 6(7):2690–2699, 2007.
- [14] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.
- [15] Ben Morris and Alistair Sinclair. Random walks on truncated cubes and sampling 0-1 knapsack solutions. *SIAM journal on computing*, 34(1):195–226, 2004.
- [16] E. Nikolova. Approximation Algorithms for Reliable Stochastic Combinatorial Optimization. *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 338–351, 2010.
- [17] E. Nikolova, J. Kelner, M. Brand, and M. Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *European Symposium on Algorithms*, pages 552–563, 2006.
- [18] Valentin Vladimirovich Petrov. On the probabilities of large deviations for sums of independent random variables. *Theory of Probability & Its Applications*, 10(2):287–298, 1965.